PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30

A1

(11) International Publication Number: WO 99/57658

(43) International Publication Date: 11 November 1999 (11.11.99)

(21) International Application Number: PCT/US99/09633

(22) International Filing Date: 3 May 1999 (03.05.99)

(30) Priority Data:
09/071,283 1 May 1998 (01.05.98) US

(71) Applicant: INFORMATION ADVANTAGE [US/US]; 7905 Golden Triangle Drive, Eden Prairie, MN 55344-7227 (US).

(72) Inventors: DRAYTON, Jay, Thomas; 13528 James Avenue South, Burnsville, MN 55337 (US). YORK, Michael; 5328 South Park Drive, Savage, MN 55378 (US).

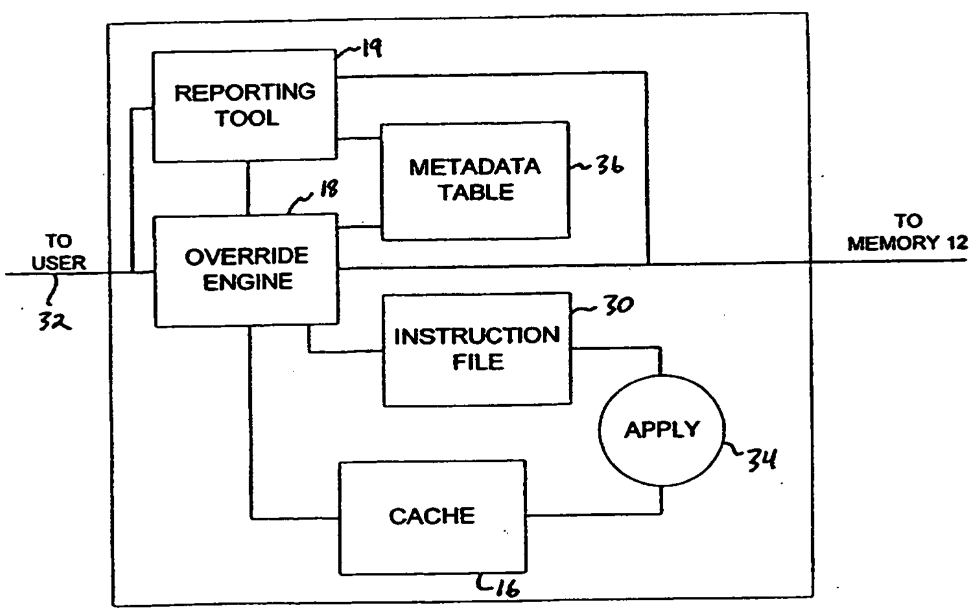
(74) Agent: VIKSNINS, Ann, S.; Schwegman, Lundberg, Woessner & Kluth, P.O. Box 2938, Minneapolis, MN 55402 (US).

(81) Designated States: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: SYSTEM AND METHOD FOR UPDATING A MULTI-DIMENSIONAL DATABASE



(57) Abstract

A data warehouse system and method. The data warehouse includes a memory and a processor. The memory includes a database having a plurality of data entries. The processor includes a cache and an override engine, wherein the cache includes a subset of the plurality of data entries and wherein the override engine extracts data from the cache for viewing by a user, modifies the data in response to one or more user commands and saves the user commands to a file for later application to the database.

SYSTEM AND METHOD FOR UPDATING A MULTI-DIMENSIONAL DATABASE

Background of the Invention

Field of the Invention

The present invention relates generally to database management, and more particularly to a system and method for manipulating facts in multi-dimensional databases.

Background Information

Business decision-makers operating in today's rapidly changing business environment need answers to a host of questions that directly impact their ability to compete in the marketplace. To manage and use information competitively, many companies are establishing decision support systems built around a data warehouse. A data warehouse stores a company's operational and historical data in an integrated relational database for decision support applications, business data access and reporting. Decision support systems access such databases to analyze and summarize corporate performance.

Data warehouses employ relational database management systems that use a language such as SQL to retrieve rows and columns of numeric data. The systems may also permit access to textual files such as documents. Data may be accessed directly via user-generated SQL commands, or indirectly, via an interface which generates the desired SQL commands. Applications such as Business Objects from Business Objects S.A., France, (http://www.businessobjects.com), Forest and Trees from Platinum Technology Inc., Oakbrook Terrace, IL, (http://www.platinum.com), and Pilot's Lightship from Pilot Software Inc., Cambridge, MA, (http://www.pilotsw.com) are typical of off-the-shelf applications which use browse windows under the control of end-users to generate the SQL code needed to analyze the data in the data warehouse.

This approach, however, runs into significant performance problems associated with PC and network limitations. Queries generated by inexperienced users can dominate and crash the database, or cause excessive network congestion. In addition, there is no mechanism for shifting large processes so they execute during off-peak hours.

OLAP (OnLine Analytic Processing) technology, also called multidimensional analysis can also be used to access the data warehouse relational database. Multidimensional analysis systems have been available for over 15 years, first on mainframes and then on client/servers. Under multidimensional analysis, data is divided into the dimensions and facts needed to manage the business. Dimensions for marketing applications may include

products, markets, distribution channels and time periods. The dimensions are used to reference specific points in a database. What that point represents is called a fact. As examples, units sold, revenue, and price are all facts. Dimensions are further described by attributes, such as size, flavor, location or fiscal year. Attributes also describe hierarchies within a dimension, even overlapping and inconsistent hierarchies. These hierarchies determine the vertical relationships within a dimension. For example, in a Period dimension, a standard hierarchy is year -> quarter -> month -> week -> day. By defining these hierarchies, it becomes possible for OLAP applications to automatically shift their 'view' up or down a hierarchy. This is commonly referred to as 'drilling' within this application space, an example of this would be shifting an annual report's total 1997 data down to view the individual quarter's numbers.

Multidimensional analysis allows users to select, summarize, calculate, format and report by dimensions and by attributes within dimensions. It can be used to support virtually any time-series decision support application including reporting, analysis, forecasting and budgeting.

To be useful, decision support systems must support analysis based not only on historical data but also on projections for future activities. For instance, marketing may project sales for the next three months. These figures may then be introduced into a model used to tune manufacturing output over that period of time. To date, such analysis has been performed using multi-dimensional databases having fixed locations. What is needed is a system and method of extracting and modifying information from an existing database which can be applied to a relational database in order to free the organization from the space limitations of multidimensional databases. In addition, what is needed is a system and method capable of creating reports not only based on existing information but also on projections of future activities.

Summary of the Invention

The present invention is a data warehouse system and method. The data warehouse includes a memory and a processor. The memory includes a database having a plurality of data entries. The processor includes a cache and an override engine, wherein the cache includes a subset of the plurality of data entries and wherein the override engine extracts data from the cache for viewing by a user, modifies the data in response to one or more user commands and saves the user commands to a file for later application to the database.

According to another aspect of the present invention, a data warehouse is described. The data warehouse includes a plurality of workstations connected to a memory by a server. The memory includes a database having a plurality of data entries. The server includes a cache and an override engine, wherein the server operates in response to user commands to store a subset of the plurality of data entries in the cache, wherein the override engine extracts data from the cache and sends the data extracted from the cache to the client workstation for viewing by a user, modifies the data in response to one or more of the user commands, saves the user commands to a file and operates in response to a commit command to modifies the database based on the user commands stored in the file.

According to yet another aspect of the present invention, a method of reporting data from a data warehouse is described in which the steps are providing a server and a memory device, storing a database in the memory device, wherein the database includes a plurality of data entries, extracting a subset of data entries from the database, storing the subset of data entries on the server, modifying the data entries stored on the server in response to user commands, reading data from the modified data entries stored on the server and displaying the data to the user.

According to yet another aspect of the present invention, a method of forecasting based on data in a data warehouse is described in which the steps are providing a server and a memory device, storing a database in the memory device, wherein the database includes a plurality of data entries, extracting a subset of data entries from the database, storing the subset of data entries on the server, modifying the data entries stored on the server in response to user commands, storing the user commands, reading data from the modified data entries stored on the server, displaying the data to the user and modifying the database based on the stored user commands.

According to yet another aspect of the present invention, a method of increasing the speed in which changes to a relational database are reflected back to the user is described in which the steps are extracting a subset of data from the database, wherein the step of extracting includes the steps of displaying a representation of the subset of data to the user and storing the subset of data in a cache, receiving a data modification command, storing the data modification command in a file and applying the data modification command against the subset of data stored in the cache, wherein the step of applying the data modification command includes the step of modifying the subset of data to reflect application of the data modification command.

Brief Description of the Drawings

In the drawings, where like numerals refer to like components throughout the several views,

Figure 1 shows a data warehouse decision support system according to the present invention;

Figure 2 shows a more detailed implementation of the data warehouse decision support system of Figure 1;

Figure 3 illustrates a star schema implementation of a data warehouse according to the present invention;

Figure 4 is a more detailed description of a data warehouse according to the present invention;

Figure 5 shows an alternate embodiment of data warehouse and decision support system according to the present invention; and

Figures 6a and 6b illustrate distribution of adjusted data across levels in an unlocked and a locked system, respectively.

Description of the Preferred Embodiments

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

Figure 1 illustrates a computer system 10 having an enhanced capacity to extract and modify data stored in a database. Computer system 10 includes a memory 12 connected to a processor 14. Processor 14 includes a cache 16, a reporting tool 19 and an override engine 18. Memory 12 is used to store a database 20. Database 20 includes a plurality of data entries 26.

In one embodiment, processor 14 also includes an instruction application process 34 for permanently applying the modifications made within override engine 18 to data entries 26 stored in database 20. In one such embodiment, database 20 is a data warehouse and override engine 18 is implemented as a multidimensional data entry/edit software engine process which supports the creation and adjustment of data points in the data warehouse.

In one embodiment, as is shown in Figure 2, cache 16 is implemented as a persistent cache which stores a subset of the plurality of data entries 26. In one such embodiment, override engine 18 extracts data from the persistent cache for viewing by a user, modifies the data in response to one or more user commands and saves the user commands to an instruction file 30 for later application to database 20.

A user sends commands to processor 14 over command interface 22 and receives reports based on data extracted from database 20 through report interface 24. In one embodiment, as is shown in Figure 2, command interface 22 and report interface 24 are implemented as a single graphical user interface (GUI) 32.

User action within system 10 is basically an adjustment process of data representing future periods in time. In one embodiment, this data will have been pregenerated by other systems and stored in the database 20. An example of this would be a statistically created forecast of future volume. A user can then "override" or adjust the values (i.e., facts) that they consider to be inaccurate. The combination of the adjustment capability with the OLAP ability to present the data in virtually any level of granularity allows business professionals to review data within a familiar business context and use their knowledge of the business to refine the data warehouse values.

As with a standard read only OLAP reporting system, System 10 allows end users and administrators to define reports to organize and present data. These reports are created by selecting the desired dimensional identifiers as well as the facts that contain the data needed to support the decision or planning process. As an example, a report may contain facts such as 'Annual Plan,' 'Statistical Forecast,' and 'Last Years Actuals' with dimensions 'Central Region,' 'Corn Syrup,' and 'May 98, June 98, and July 98.'

All the data the user sees on the report is stored in a work file that was created for that adjustment cycle. The process administrator creates one work file for each adjustment cycle. For example, in a monthly planning cycle there would be one work file for the May 1996 cycle, one for the June 1996 cycle, and so on.

When the user adjusts a fact value, the information is saved in a report file, not the work file. In the embodiment, when the final adjustment of the fact is completed, the process administrator updates the database with the new forecast data.

As an example, the following chart summarizes a typical adjustment cycle. In this case, a cycle used to create a consensus tactical forecast is shown. The chart identifies the required input, the steps, and the results.

Override Process: Tactical Forecast Cycle		
Input to the Cycle	**Steps of the Cycle**	**Results of the Cycle**
Statistical forecast data, tactical work file, tactical fact.	1. Users create reports using filters and dimensions. They work *only* with the assigned products/ markets/periods.	Preliminary reports.
	2. Users adjust last month's tactical fact to create this month's tactical forecast. They save the forecast amounts, and distribute the forecasts for review.	Tactical forecast reports for others to review, work with, and change.
	3. Users make final changes and save the tactical forecasts.	Final version of tactical forecasts.
Report files, tactical work file.	4. Forecasting administrator completes the cycle.	Management work file, management fact.

As an additional example in which the override function is used in a forecasting application, the following chart summarizes a subjective management forecasting cycle. It identifies the required input, the steps, and the results.

Override Process: Management Forecast Cycle		
Input to the Cycle	**Steps of the Cycle**	**Results of the Cycle**
Management work file, management fact.	1. Users create reports using filters and dimensions. They work *only* with the assigned products/ markets/periods.	Preliminary reports.
	2. Users adjust last month's management fact to create this month's management forecast. They save forecast amounts, and distribute the forecast for review.	Reports for others to review, work with, and change.
	3. Users make final changes and save the management forecast.	Final version of management forecasts.
Report files, management work file.	4. Forecasting administrator completes the cycle.	Final business fact.

At the end of each cycle, the process administrator prepares the data for the next cycle. This involves saving the data to the database, preparing the work file for the next cycle, and setting up the single adjustable fact for that cycle. The adjustable fact is the fact that can be changed or overridden.

For example, at the end of the tactical forecast cycle, the Forecasting administrator updates the database with the consensus tactical forecast and sets up the next planning cycle, in this case a consensus management forecast. The following chart lists the steps.

Input to the Cycle	Steps taken by the Forecasting administrator	Results of the Cycle
Work file for the cycle (tactical work file, and management work file).	1. Appends all report files from all users. 2. Commits the data to the Forecasting database. 3. Sets the fact to be adjusted in the next forecasting cycle.	Integration of forecast rules. Updated Forecasting database. A new adjustable fact for the next cycle.

A forecast override application such as system 10 allows users to apply qualitative methods (i.e., business knowledge) to the forecasting process. Users can change values generated by purely statistical methods to reflect changing business conditions.

In one embodiment, system 10 maintains a master activity file. The master activity file consists of the full set of dimension information user information, and security information. It also contains detailed information about all of the facts that were chosen by the process administrator to support the planning or adjustment process. It is stored in only one place.

Individual users have their own configuration files and the appropriate fact data files. When the override application is started, the master data file is read and the entire dimension class is built in memory for the user. The data structure has pointers to all of the data that the master structure points to. A system end user can view and modify any data that he has permission to see. The data that they are allowed to see can be configured on a user by user basis to ensure that they see data they are authorized to access and in a usable context.

A lock file is created while the work file is being processed by a user to ensure atomic usage of the data in the file.

Reports

In one embodiment, reports are created from a template. The template is like a blueprint for the report. Each time a report is created, system 10 saves the blueprint.

In one embodiment, report templates are shared among users. Each user can then create their own version of the report. For example, they can run a new report, modify the

report template, or "drill" to create a different version of the report. By drilling, they are able to review and adjust the data anew, with more discreet levels of detail. As an example, instead of simply adjusting the fact 'Planned Budget' up by 25% at the annual level, the end user may 'drill down' to the monthly level and review and adjust specific months values.

All reports contain one or more columns and one or more rows. Each report can consist of multiple sections. If a report contains multiple sections, each section contains the same type of row and column information.

The section, column, and row names in a report reflect the dimensions of database 20. These dimensions give meaning to the values in database 20. It is, therefore, important to understand how the dimensions work together.

Every data value in database 20 is defined by one or more dimensions. To fully describe each value, a report must contain at least one element from each of the available dimensions.

Period dimensions are time intervals used for identifying and consolidating the data, such as weekly, monthly, quarterly, and yearly intervals. Non-period dimensions describe other aspects of your data. Non-period dimensions may include, for example, *Geo-Political*, *Product*, and *Business_Org*.

Each dimension can consist of hierarchical levels. For example, *product* may be a dimension. Brand and SKU are associated product dimension levels. The levels represent differing degrees of detail and the paths used when you drill.

In one embodiment, database 20 is a data warehouse stored in one or more work files. Each work file can contain an unlimited number of data points. (That is, the work file size is limited only by the amount of physical storage.) In one such embodiment database 20 is represented as a multidimensional database. Override engine 18 can modify and view data points at any combination of levels within the multidimensional database and, as values are changed, the new values are allocated to all levels of the dimension hierarchies. This ensures that summary levels of the data still total correctly. User instructions to the override engine 18 are not applied to the data points. Instead the instructions are stored in instruction file 30. Since a single instruction may change tens of thousands of rows, just storing the instructions is much faster and less expensive than storing all the changed values. Once editing is complete the edits can be applied to the work file by executing instruction application process 34. In this way, large scale data warehouse updates can be performed offline while the user is working on other tasks.

By storing the instructions rather than automatically applying them, computer system 10 can present the user with the results of the desired changes more quickly than in previous data warehouse analysis products. In addition, by using one or more work files rather than a predefined multidimensional database (MDDB) product, system 10 can handle data warehouses which are larger in size than current database products.

In one embodiment, override engine 18 extracts the subset of data entries 26 from database 20 and stores the extracted data into cache 16. In one such embodiment, override engine 18 employs the same mechanism used by reporting tool 19 to extract data from database 20. For instance, in one such embodiment, both override engine 18 and reporting tool 19 determine the appropriate levels of data to extract from database 20 by querying an OLAP object running on a standard request broker. Similarly, both override engine 18 and reporting tool 19 look to a Metadata table such as Metadata table 36 to identify fact tables, determine the levels they need in each fact table, and to extract the data from each required fact table. Override engine 18 then takes the report generated and pushes it into cache 16. From that point on, override engine 18 receives a command, executes the command, and presents the data to the user. In addition, override engine 18 stores the command in instruction file 30 so that if one wants to rerun that report later, all that has to happen is that the commands are reapplied to the known starting point.

As noted above, override engine 18 provides the user or users with a mechanism for quickly adjusting data stored in the data warehouse during a subjective fact adjustment session. In one embodiment, data stored in the data warehouse can be in any of three states during such an adjustment cycle. Data that came out of database 20 is "untouched." Commands to alter the data are stored to instruction file 34 and are used to change the data being displayed to the user. The commands do not, however, change the data in either cache 16 or database 20.

Data that's been stored back to the data warehouse (i.e., stored in database 20) is "adjusted" or "written" data. Data in database 20 is modified using the instructions stored in instruction file 34 via a "commit" command. In one such embodiment, data stored in persistent cache 16 is invalidated during execution of the "commit" command.

Data which has been adjusted in cache 16 but which has not yet been written to database 20 is in that more nebulous state in the middle where it is adjusted but not committed. And that's the situation where cache 16 does not match what's in the data warehouse anymore. In one such embodiment, the data gets to that state by executing a

"release" command in engine 18. The "release" command causes override engine 18 to apply the commands not only to the data presented to the screen but also to the data within cache 16 itself. In one such embodiment, commands must be "released" before the resulting changes are "committed" to database 20. Once a set of commands are "released", however, the original data is gone.

In another embodiment, override engine 18 maintains a stack of old states and the user can scroll back through a list of releases to recover an older state of cache 16.

In one embodiment, system 10 only extracts data from database 20 at the start of the forecasting process. In such an embodiment, system 10 only extracts data from database 20 at the start of the forecasting process. In such an embodiment, System 10 applies a three-step process: work file generation, interactive overriding, and database storage. Work file generation is accomplished using the extract phase. Extract reads the base statistical forecast data from database 20 and stores the data in a Work File (cache 16) in the UNIX file system of processor 14.

The remaining phases (override, release, and commit) use the data in the Work file to generate adjusted fact data. System 10 allows the user to interactively change individual product or market values to reflect changing business conditions. Each adjustment is allocated up and down the dimension hierarchy (aggregation levels). Allocations are performed by using the dimension's drill hierarchy to identify lower level components of the data point adjusted. These lower level components are then given the new value that maintains their relative contribution to adjusted value. Users may also lock or unlock individual values so that the values do not change during the adjustment process (directly or indirectly).

Once the adjustor has completed all work for an override, the adjustment is checked-in to the work file. The check-in process ensures that two adjustors do not adjust the same values and, in one embodiment, it allows a process administrator to review the work before commit.

The final phase of the override process is the database storage phase, or commit. Once the user or administrator is satisfied with the generated overrides, the overrides must be committed to the database. This is accomplished in the storage phase.

As noted above, extraction is the first phase of the override process. It is the generation of the work file. Extraction builds a snapshot of database 20 and stores the snapshot in the Work File in the UNIX file system of processor 14. Extraction is generally

done once because it is a time-consuming process and you generally do not want or need a user to see that happen. In one embodiment, the extraction process is performed by an agent executing within reporting tool 19. The agent will typically be set up to perform the extraction process after database 20 is refreshed.

In one embodiment, the extraction agent operates in response to user commands to extract the correct level of data from database 20. In one such embodiment, one could set up system 10 to respond to a command such as "I want these markets by these products for this period on the horizon, by channel" (or whatever your other dimensions are). Another command syntax might be "Give me all my products and give me all my markets at the region level over a selected forecast horizon" (e.g., current to current-plus-12).

In one such embodiment, refresh of the persistent cache is triggered by the current flag moving forward one month. For example, if the current month switches from January to February, a subset of data is read from database 20 and a combination of facts from database 20 and calculated facts formed as a function of data in database 20 are stored in cache 16. Any data resident in persistent cache 16 at the time of the refresh that is unsaved or uncommitted is simply overwritten.

To build the Work File, the system 10 must determine which dimensions are drillable and at which aggregation levels the non-drillable dimensions will be forecast. In one embodiment, this information is stored in the Category table of the Metadata.

For drillable dimensions, data is stored in the Work File at the lowest level defined in the drill hierarchy for that dimension. For non-drillable dimensions, data is stored only at the level of aggregation specified in the Metadata for that dimension.

The Override phase allows the user to interactively adjust the base statistical forecast data. Data is read from the Work File generated by the extraction phase. The user is allowed to adjust values, lock and unlock values, drill up and down the drillable dimension's hierarchy, as well as many other functions.

Once an adjustor has completed a set of override changes, these changes are released to the work file so that they are available for final commit to the warehouse. The release phase loads the base data from the work file, loads an instance file which contains the saved set of changes, and writes the new values back to the work file.

The commit phase performs the commit for the Override process. First, it reads the Work File. Next, the data values of the lowest-level decomposed data are read from the warehouse. These values are used to properly proportion the data as it is decomposed to the

lowest level in each dimension's hierarchy. Finally, the adjusted and decomposed data is written to the database.

For example, suppose the data in the Work File is stored at the regional (REG) level of the Market dimension. Further suppose that the lowest level in the Market hierarchy is market (MKT). Store will read the REG level data from the work file, decompose the REG level data to the MKT level, and finally write the MKT level data to the database.

The Database Structure

In one embodiment, database 20 is implemented as a non-normalized star schema. A simplified version of a star schema 38 is shown generally in Fig. 3. In a star schema, facts are stored as data in fact tables. The fact tables are indexed by a multi-part key made up of the individual keys of each dimension. Similarly, dimension information is stored in dimension tables. In the embodiment shown in Fig. 4, DimTable 40 is a dimension table while the tables labeled "Warehouse Product", "Warehouse Market", "Warehouse Period" and "Warehouse Fact" are fact tables 42, 44, 46 and 48, respectively.

Non-normalized star schemas are designed for very fast data aggregation and calculation. Such speed can be very advantageous in building the subset of data to be stored in cache 16. Such systems can, however, bog down considerably when required to perform incremental updates (e.g., as is the case during a forecasting session). That is where the use of cache 16 and instruction file 30 are instrumental in supporting such forecasting sessions.

The downside of a star schema approach is that you end up with widespread replication of data across tables. At the same time, however, this replication of data give you the ability to get your keys from the data warehouse with very small queries that database 20 certainly can handle quickly. So a star schema approach is very well optimized for queries that pull a large number of rows out of database 20.

Standard OLAP reporting tools do not have to understand multiple levels within a product or market hierarchy of a multi-tiered data warehouse. This is a key difference between override engine 18 and a standard OLAP reporting tool. Override engine 18 must understand multiple levels at the same time because for the purposes of an adjustment, there may be interdependencies on the levels themselves for a reporting tool that essentially say, "O.K., here's data at this level," or "Here's data at the combination of these two levels." They're largely independent of each other and you can concentrate on one or the other at any given time. Because of that, override engine 18 does not run directly against the warehouse.

Instead, override engine extracts that data, performs calculations where necessary and stores the modified data to cache 16.

In order to operate correctly, however, with database 20, override engine 18 and instruction application process 34 must understand the structure of database 20. In one embodiment, as is shown in Figure 3, override engine 18 and instruction application process 34 extract the structure of database 20 by reviewing the contents of Metadata table 36.

Such an approach permits the use of an unlimited and easily alterable number of dimensions. In its simplest form, metadata table 36 includes a period dimension table 40 and a fact table 46. In one embodiment, each dimension table includes a unique key, unique description field, a level column and a hierarchy level column. In addition, period dimension table includes a unique sequence number column, a sequence within year column and a current period column. Fact table 46 includes keys which are identical in type and structure to the keys listed in dimension table 40.

For example, the metadata table can be used to drive a drill hierarchy that tells database 20 that days make up weeks and weeks make up months and months make up quarters, quarters make up halves, halves make up years. Or that four quarters make up a year, so that the user can jump and skip things. Same thing in the product hierarchy. Override engine 18 and reporting engine 19 understand the Metadata structure and use the data stored in the dimension tables to extract data from database 20.

For example, a certain table may include a category of sales by total U.S. And maybe it is category, manufacturer, brand. And then over in another table are lists of SKUs by store.

Another approach is to have a separate fact table for each month's data. That way when the next month's data arrive, it gets placed into a new fact table and stored in database 20. An advantage of such an approach is that if one of the fact tables gets lost or corrupted, you can reload that month. In addition, as database 20 gets bigger and bigger, its value to the company increases and so does the cost of maintaining the database. Relational databases can scale into much larger data sizes and are much more maintainable than a corresponding multi-dimensional database.

It should be understood that database 20 may be distributed across a number of computers. In one embodiment, such as is shown in Fig. 5, computer system 50 includes a server 52 connected to a processor 58 and a processor 62 by a network 56. In addition, server 52 is connected to a plurality 1 through N of workstations 62. Processor 58 and processor 62 store portions of database 20 in memories 60 and 64, respectively.

In one embodiment, as is shown in Fig. 5, server 52 maintains a separate instantiation 54 of override engine 18, reporting tool 19 and cache 16 for each user performing the forecasting function. In another embodiment (not shown), a single cache 16 is used across all users. In either case, changes to cache 16 are maintained in a separate instruction file 30 for each instantiation 52 and are only applied to the cache on receipt of a "release" command.

In one embodiment, processor 58 is a multiple processor machine such as the Hewlett Packard HP9000 running an Oracle database application while processor 62 is a Tandem Himalaya 128. In one such embodiment, aggregated data is stored in the Oracle database on the HP machine while the lowest level data is stored in the Himalaya machine. The Metadata tells the program the appropriate processor to which a query should be addressed. For example, if data is stored by category by store in processor 62, override engine 18 can determine this be looking at the Metadata. If, on the other hand, the user is storing aggregate data in cache 16, it will determine from the Metadata table that it should extract such data from the Oracle database on processor 58.

In another embodiment, override engine 18 and reporting tool 19 run as a relatively thin windows client that essentially just provides GUI 32. All the analytics, all the storage, and all the real processing is done on UNIX server 52.

Locking Values in the Database

In one embodiment, override engine includes the ability to lock values in database 20. As noted above, on a release, the changes at one level of the hierarchy are pushed down to each of the sublevels of the hierarchy. For example, if production is increased across the board by 10,000 units, the 10,000 units are distributed proportionally across each of the entities at the lowest level of the hierarchy.

In certain situations, such an approach does not make sense. For instance, if one of three manufacturing plants is operating at capacity, it makes little sense to distribute the increase proportionally to the plant operating at capacity. For such situations, override engine 18 includes a locking mechanism which can be used to lock the output of any of the manufacturing plants to a certain value. Any increases are then distributed proportionally across the remaining, unlocked, manufacturing plants.

In one embodiment, the adjustment report displays the values of the dimensions the user selects on a Template dialog screen. The only values that can be adjusted for the

forecast are found in the unshaded cells. Locked values are shaded red and non-adjustable values are shaded yellow.

The user can adjust a value at any level and the effects ripple through the rest of the product hierarchy. For example, if he or she adjusts a SKU value, the values of products at higher levels change to account for the lower level adjustment. Or, if the user adjusts a higher level value, the values for the lower level products change as needed.

Likewise, when the user locks a value, product values both up and down the hierarchy may be affected. For example, if the top level is locked at 100, all lower values must total 100.

The diagram shown in Fig. 6a is an example of decomposition or proportional fitting based on current forecast data. The current adjustment data is retrieved at manufacture level 70 and, after an adjustment, is proportionally propagated through Brand level 72 to SKU level 74.

If, however, the manufacture product value was locked at level 70 and one Brand product value was adjusted from 50 to 60, the adjustment and its effects are shown in Fig. 6b.

As noted above, in one embodiment both reporting tool 19 and override engine 18 have the ability to limit the effect of commands such that the data returned from database 20 does not include the entire hierarchy for all dimensions. For instance, one may care about multiple levels of product dimension, but would like to limit the market adjustments to a particular level (e.g., make the market adjustments at a total U.S.). In one such embodiment, the user can select the level at which data is adjusted. For example, one would tell override engine 18 to, "Make your product dimension drillable, make your market dimension non-drillable, make your period dimension non-drillable." Adjustments then must only be driven down the product hierarchy for display; the override engine does not have to try to drive it down the other hierarchies. As you add multiple drillable dimensions, the amount of work that has to happen when you make adjustments expands exponentially. Not only does it go down product, but it has to go down product for every market, or every submarket, or every submarket of every submarket and you end up with a huge matrix of numbers to keep track of.

Drilling allows the user to display the fact values at different product levels, both up and down the product hierarchy. With drilling, one can see the aggregate values of a group of products, or can identify the specific values that went into the aggregate amount.

Drilling up works with any product where there are other products at a higher level. For example, one can drill up on a value if he or she is at SKU level 74 shown in Fig. 6a.

Finally, a user can drill down with any product if he or she is at a level above the lowest level, SKU. For example, one can drill down if at manufacture level 70 in Fig. 6a.

If, for instance, you want to adjust the amount of a particular soft drink sold in a particular city, you could either make that a drillable thing or a non-drillable dimension. You can do it in one of two ways. One is you can set your market dimension as drillable, and then poll the whole market hierarchy in order to drill down to the city and make the change. If, however, you did not need that flexibility in drilling and the associated performance impact of allocating the changes through that dimension, you could make the market dimension 'non-drillable' and only see data at the city level. Then you can just grab whatever city you wanted and make your adjustments. But because you probably have the data stored at some level below city, once you do a "commit" it is going to drive the numbers to the bottom of all the hierarchies. (In one embodiment, override engine drives the numbers down to the bottom of all the hierarchies by modifying each number as a percent of contribution to the total.)

For example, if given a market hierarchy including regions, cities within those regions and stores within those cities, you were going to modify units at the regional level, you only need to maintain data in cache 16 at that level. To do this, override engine 18 initiates a transfer of data at the atomic level, receives the data and pushes it up to the regional level. The aggregated regional data is then stored in cache 16 and commands from the user are applied against that aggregate data. Then, when you are ready to store the changed data, a "commit" command is executed and the modifications get drilled down to the atomic level in database 20. The result is that user commands get executed quickly at the regional level and get stored accurately at the atomic level.

For example, one might determine what would happen if volume increased by 10% in a region such as the Great Lakes territory. The change would be made at the territory level and, when committed, it is distributed to the cities as a function of the percentage they contribute to the regional total.

No matter the level where the adjustment is made, once the warehouse is updated, it is updated at the lowest atomic level and then in this case for reporting engine 19, then it can automatically pick up a report of it based on market hierarchy, period hierarchy, etc. All that is a trade-off between speed and flexibility. You can go ultimately flexible if you've got enough iron behind the thing to drive it. And if you do not, then you can cut back.

Also from a security perspective, quite often if it is sales people that are using it for the adjustments, they do not want them to see other territories, for instance, other people's clients, other sales guys' clients. Otherwise they start playing games that you do not want them playing. It is a combination of those things.

Override engine 18 understands all of reporting tool 19's filters and calculated facts; it also understands the hierarchies of system 10. Essentially everything in the Metadata that can be defined, is read by reporting tool 19 and override engine 18 and they will use those definition. So you can create a calculated fact in reporting tool 19 and override engine 18 will automatically see it and know how to use it.

(An example of a calculated fact would be gross revenue. Gross revenue is calculated as units times price. You would not, therefore store gross revenue. Instead, you calculate it off of the two that you did store (price and units). Another calculated fact is forecast error. Facts calculated as a function of an actual fact and a forecasted fact are derived, not stored. Override engine 18 automatically reads the definition of each calculated fact out of the Metadata and applies them on the fly to the data read from database 20.)

In one embodiment, forecasts are stored as separate databases within database 20. For instance, one could include a dimension labeled "scenario" into the warehouse. It can, therefore, be critical to have the ability to add extra dimensions to systems 10 and 50.

Cache

As was noted in connection with Figure 2, in one embodiment cache 16 is a persistent cache stored as a B-tree in a UNIX file system on processor 14. The B-tree allows you to essentially go with a much more compact data storage in situations where the matrix of data is populated sparsely (such as in, for example, customer centric databases) yet, at the same time, a B-tree implementation does allow you to get reasonably fast access times.

In one such embodiment, the persistent cache is implemented using standard Rogue Wave B-tree code. The performance of the Rogue Wave B-tree code can be enhanced by replacing the standard I/O stream (which is write character by character) with an I/O stream which writes data in large blocks of data.

In addition, with a B-tree implementation, one can trade off size for speed by increasing or decreasing the number of branches and subbranches in the tree. That is, the deeper you make the B-tree, the slower your access time but, at the same time, the smaller the

actual physical storage. In one embodiment, the structure of the B-tree is optimized for size over speed.

In more densely packed matrices of data, cache 16 can be implemented as a set of file-based arrays. Such arrays are very fast, requiring only simple calculations to get to the right locations. They can be, however, very inefficient space-wise. Typically, if one of the dimensions of the data warehouse is customer, and the number of potential customers is large, you are better off selecting a B-tree implementation for cache 16.

Override engine 18 allows the user to grab slices of data from a database that may be too big to comprehend in its entirety. Override engine 18 allows you to slice a portion of the database out, which sometimes in the industry they'll refer to as a datamark, automatically and intelligently through the user of metadata. The portion extracted can be dealt with either within or outside the database quickly and efficiently. Once the changes are in place, override engine 18 automatically and seamlessly pushes them back into database 20. The result is a database tool which eliminates the query traffic bottleneck of traditional approaches to relational database management systems.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiment shown. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is intended that this invention be limited only by the claims and the equivalents thereof.

What is claimed is:

1. A data warehouse, comprising:

memory, wherein the memory includes a database having a plurality of data entries;

a processor connected to the memory, wherein the processor includes:

a cache; and

an override engine;

wherein the cache includes a subset of the plurality of data entries and wherein the override engine extracts data from the cache for viewing by a user, modifies the data in response to one or more user commands and saves the user commands to a file for later application to the database.

2. A data warehouse, comprising:

memory, wherein the memory includes a database having a plurality of data entries;

a server connected to the memory, wherein the processor includes:

a cache; and

an override engine; and

a client workstation connected to the server;

wherein the server operates in response to user commands to store a subset of the plurality of data entries in the cache, wherein the override engine extracts data from the cache and sends the data extracted from the cache to the client workstation for viewing by a user, modifies the data in response to one or more of the user commands, saves the user commands to a file and operates in response to a commit command to modifies the database based on the user commands stored in the file.

3. A method of reporting data from a data warehouse, the method comprising the steps of:

providing a server and a memory device;

storing a database in the memory device, wherein the database includes a plurality of data entries;

extracting a subset of data entries from the database;

storing the subset of data entries on the server;

modifying the data entries stored on the server in response to user commands;

reading data from the modified data entries stored on the server; and

displaying the data to the user.

4. A method of forecasting, comprising the steps of:

providing a server and a memory device;

storing a database in the memory device, wherein the database includes a plurality of data entries;

extracting a subset of data entries from the database;

storing the subset of data entries on the server;

modifying the data entries stored on the server in response to user commands;

storing the user commands;

reading data from the modified data entries stored on the server;

displaying the data to the user; and

modifying the database based on the stored user commands.

5. A method of increasing the speed in which changes to a relational database are reflected back to the user, comprising the steps of:

extracting a subset of data from the database, wherein the step of extracting includes the steps of:

displaying a representation of the subset of data to the user; and

storing the subset of data in a cache;

receiving a data modification command;

storing the data modification command in a file; and

applying the data modification command against the subset of data stored in the cache, wherein the step of applying the data modification command includes the step of modifying the subset of data to reflect application of the data modification command.

6. The method according to claim 5, wherein the step of applying the data modification command further includes the step of modifying the representation of the subset of data to reflect application of the command.

7. The method according to claim 6, wherein the step of applying the command further includes the steps of:

waiting for a "commit" command; and

on receipt of the "commit" command, modifying data in the database to reflect application of the data modification command.

8. The method according to claim 5, wherein the step of applying the command further includes the steps of:

waiting for a "commit" command; and

on receipt of the "commit" command, modifying data in the database to reflect application of the data modification command.

9. A data structure for updating a database, comprising:

a plurality of user data-modification commands, wherein the plurality of user data-modification commands provide information for modifying information in a database (20).

10. The data structure of claim 8, wherein:

the plurality of user data-modification commands provide data-modification information for application to aggregated regional data stored in a cache (16), and modification information that gets drilled down to an atomic level in the database (20).

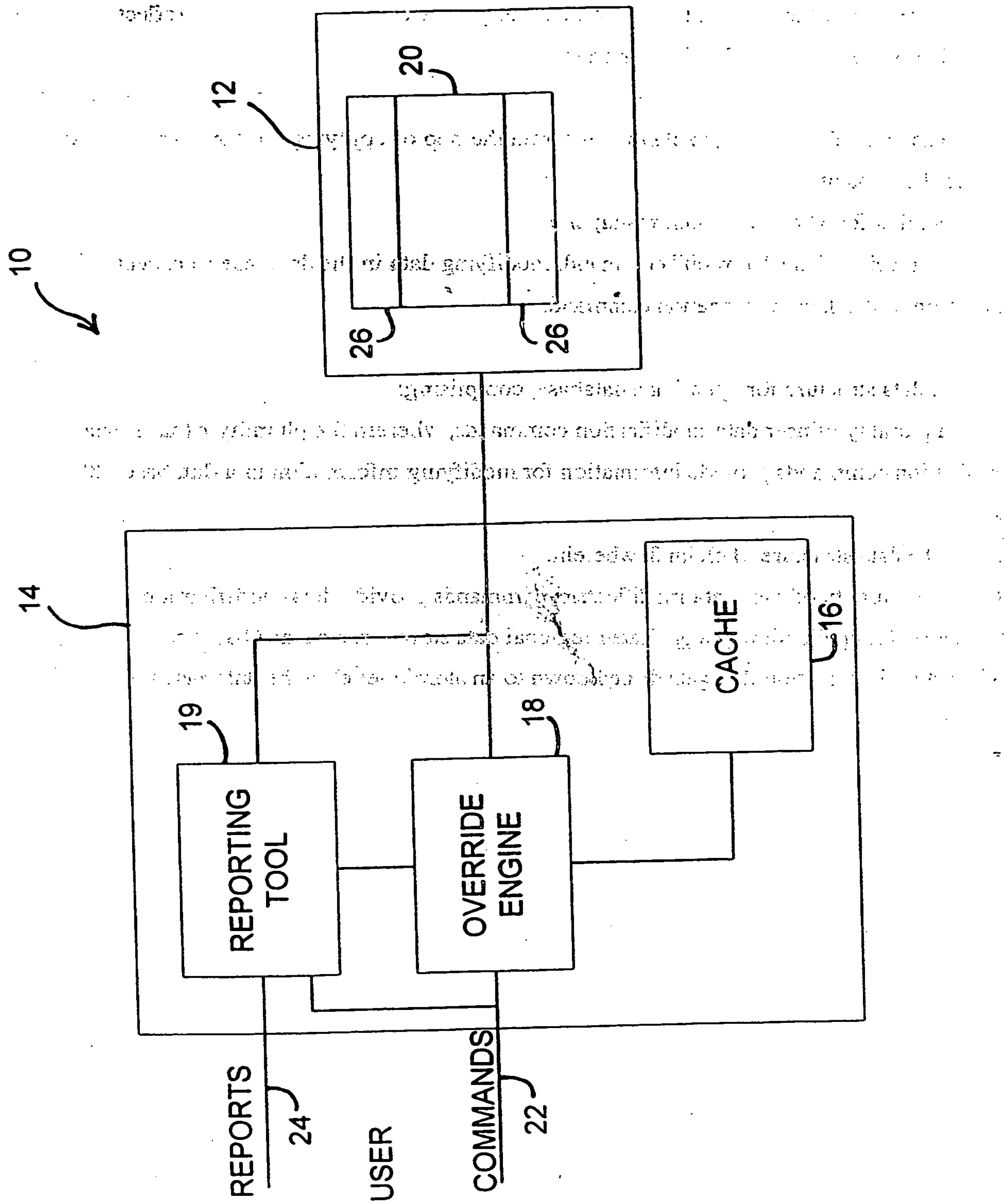

10
12
20
26
26
14
19
18
16
CACHE
REPORTING TOOL
OVERRIDE ENGINE
REPORTS
24
USER
COMMANDS
22

Fig. 1

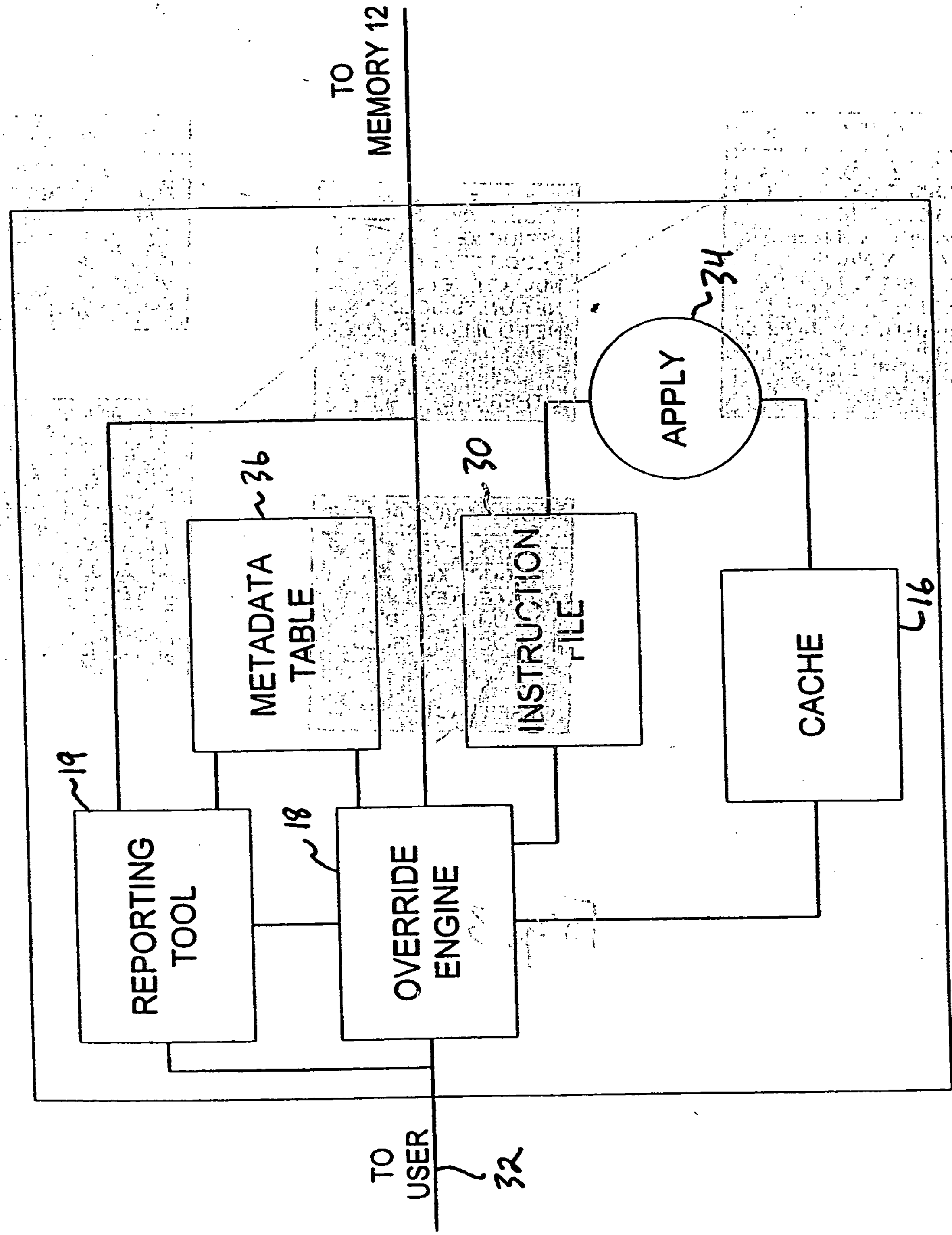

TO MEMORY 12
APPLY
34
METADATA TABLE
36
INSTRUCTION FILE
30
CACHE
16
REPORTING TOOL
19
OVERRIDE ENGINE
18
TO USER
32

Fig. 2

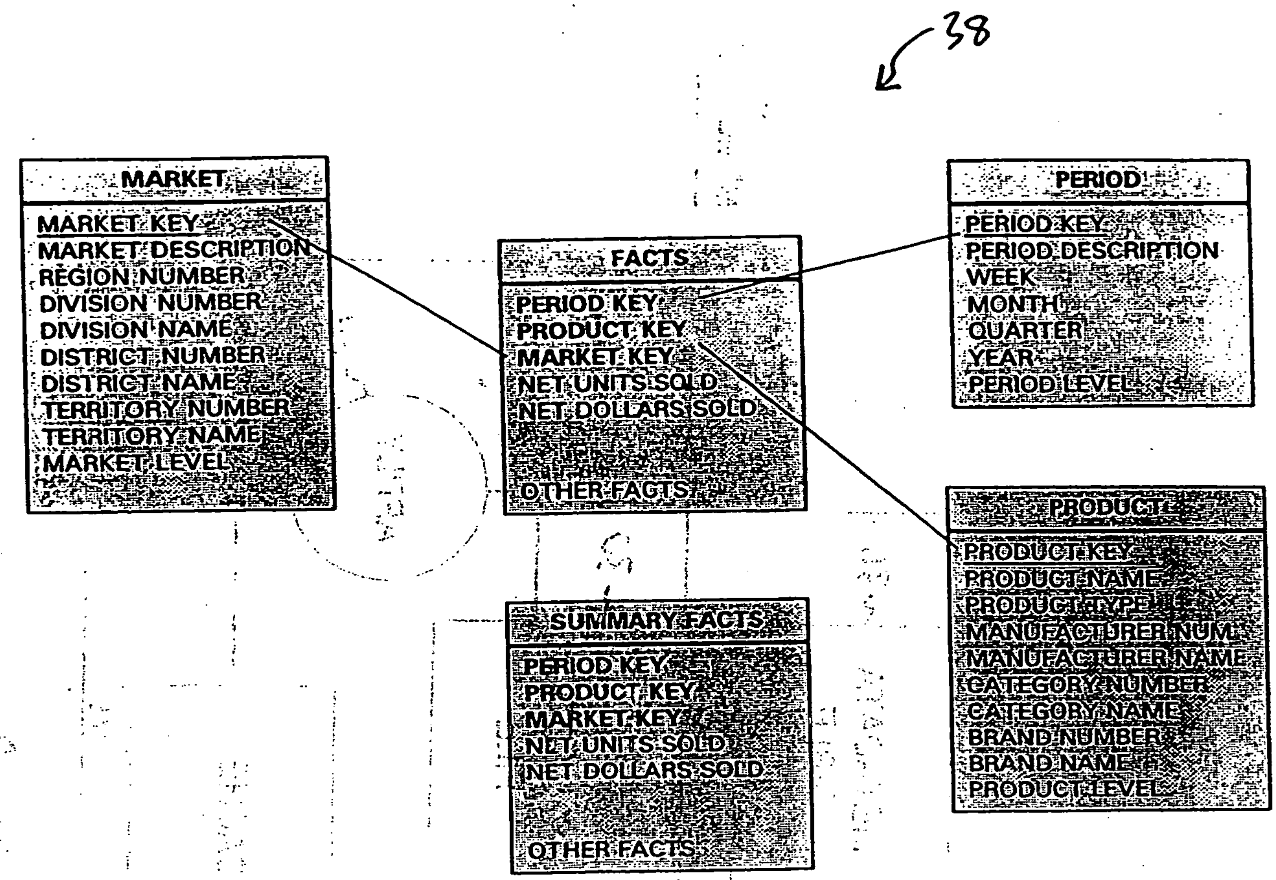
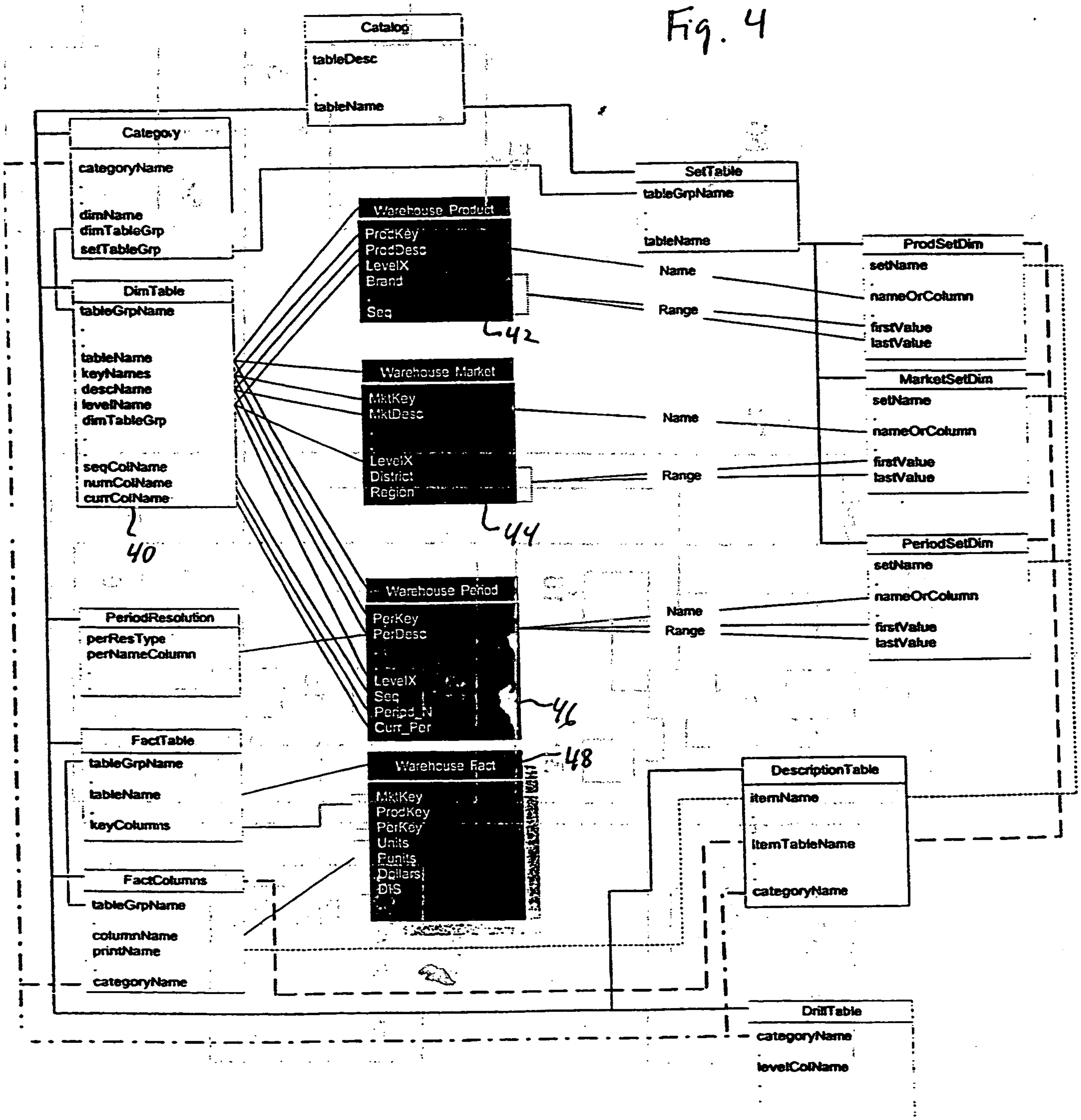

38
MARKET
MARKET KEY
MARKET DESCRIPTION
REGION NUMBER
DIVISION NUMBER
DIVISION NAME
DISTRICT NUMBER
DISTRICT NAME
TERRITORY NUMBER
TERRITORY NAME
MARKET LEVEL
FACTS
PERIOD KEY
PRODUCT KEY
MARKET KEY
NET UNITS SOLD
NET DOLLARS SOLD
OTHER FACTS
PERIOD
PERIOD KEY
PERIOD DESCRIPTION
WEEK
MONTH
QUARTER
YEAR
PERIOD LEVEL
PRODUCT
PRODUCT KEY
PRODUCT NAME
PRODUCT TYPE
MANUFACTURER NUM
MANUFACTURER NAME
CATEGORY NUMBER
CATEGORY NAME
BRAND NUMBER
BRAND NAME
PRODUCT LEVEL
SUMMARY FACTS
PERIOD KEY
PRODUCT KEY
MARKET KEY
NET UNITS SOLD
NET DOLLARS SOLD
OTHER FACTS

Fig. 3

Fig. 4


Catalog
tableDesc
tableName
Category
categoryName
dimName
dimTableGrp
setTableGrp
DimTable
tableGrpName
tableName
keyNames
descName
levelName
dimTableGrp
seqColName
numColName
currColName
40
SetTable
tableGrpName
tableName
Warehouse Product
ProdKey
ProdDesc
LevelX
Brand
Seq
42
Name
Range
ProdSetDim
setName
nameOrColumn
firstValue
lastValue
Warehouse Market
MktKey
MktDesc
LevelX
District
Region
44
Name
Range
MarketSetDim
setName
nameOrColumn
firstValue
lastValue
PeriodSetDim
setName
nameOrColumn
firstValue
lastValue
Warehouse Period
PerKey
PerDesc
LevelX
Seq
Period_N
Curr_Per
46
Name
Range
PeriodResolution
perResType
perNameColumn
FactTable
tableGrpName
tableName
keyColumns
Warehouse Fact
48
MktKey
ProdKey
PerKey
Units
Eunits
Dollars
DIS
FactColumns
tableGrpName
columnName
printName
categoryName
DescriptionTable
itemName
itemTableName
categoryName
DrillTable
categoryName
levelColName

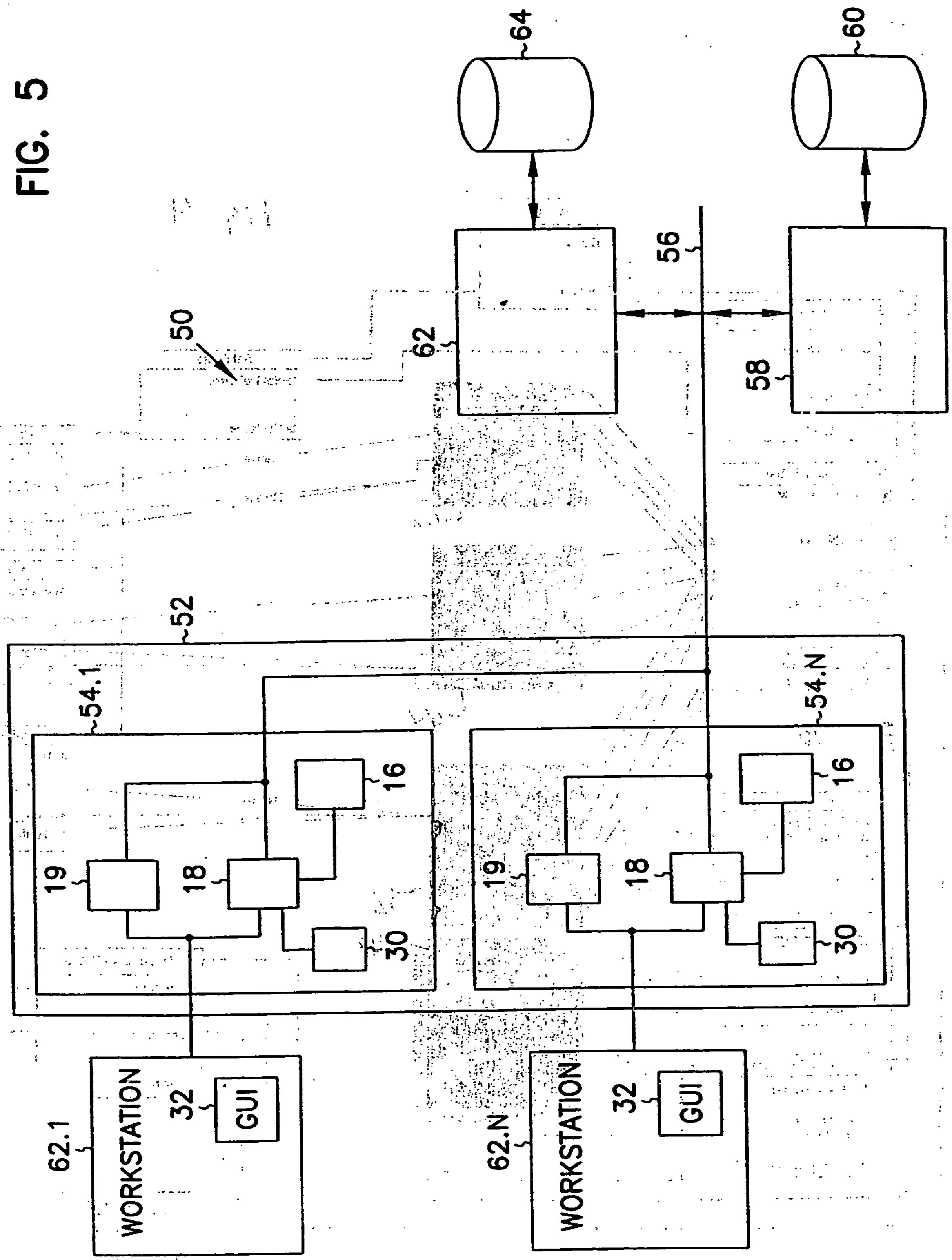

50
52
54.1
54.N
56
58
60
62
64
16
18
19
30
WORKSTATION
GUI
32
62.1
62.N

FIG. 5

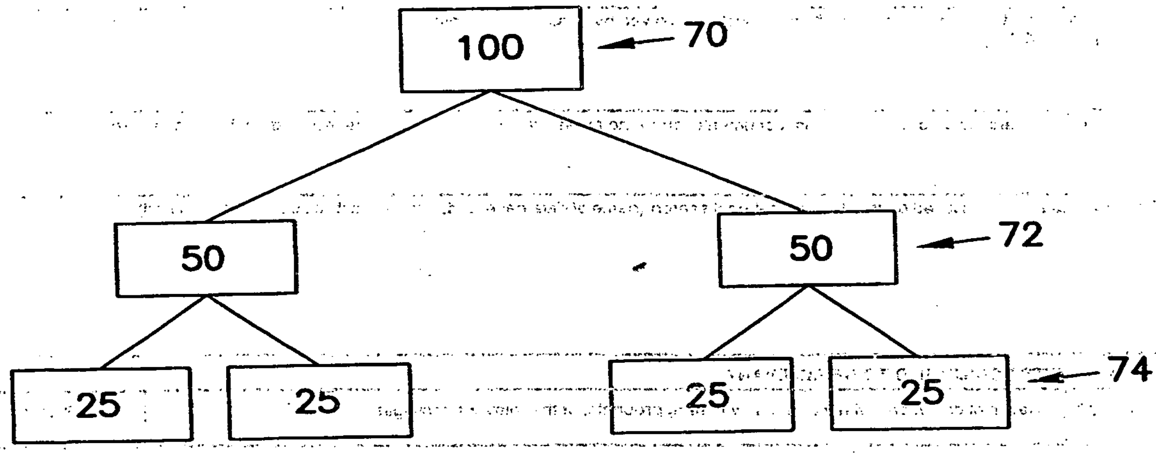

100
70
50
50
72
25
25
25
25
74

FIG. 6a

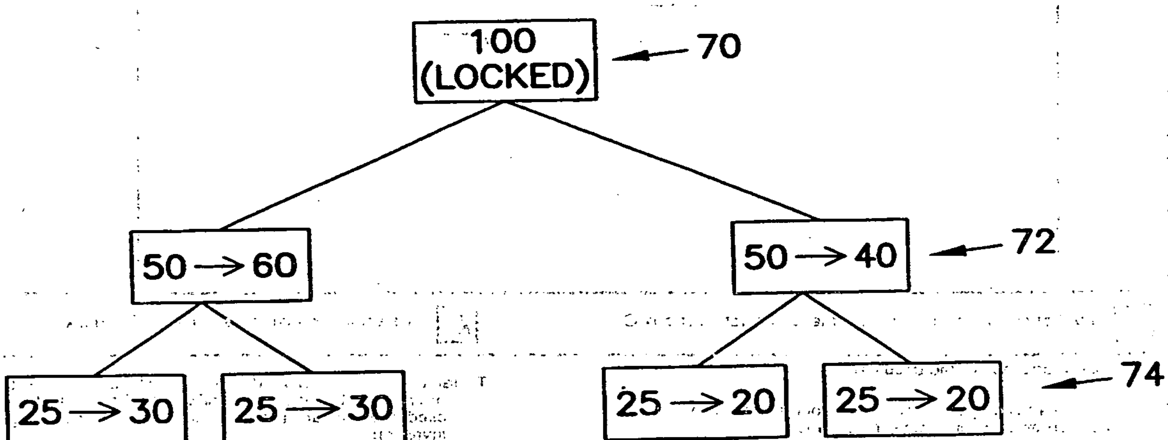

100
(LOCKED)
70
50 → 60
50 → 40
72
25 → 30
25 → 30
25 → 20
25 → 20
74

FIG. 6b

INTERNATIONAL SEARCH REPORT

Interr ational Application No
PCT/US 99/09633

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"Oracle announces Beta of Oracle Discoverer 3.1" BITECOMM RELEASES, 'Online! 26 March 1998 (1998-03-26), pages 1-2, XP002115172 Retrieved from the Internet: <URL:http://www.bitecomm.co.uk/releases/oracle/98-2-26-18897.html> 'retrieved on 1999-09-13! the whole document ---	1-5

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 September 1999	27/09/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Fournier, C

1

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	CHAUDHURI S ET AL: "An overview of data warehousing and OLAP technology" SIGMOD RECORD, MARCH 1997, ACM, USA, vol. 26, no. 1, pages 65-74, XP002115173 ISSN: 0163-5808 abstract; figure 1 page 70, left-hand column, paragraph 6 -page 72, right-hand column, line 4	1-5
A	WO 98 09238 A (AT & T CORP) 5 March 1998 (1998-03-05) page 1, line 5 -page 7, line 13	1-5,9,10
X	EP 0 420 419 A (HITACHI LTD) 3 April 1991 (1991-04-03) abstract	9
A	US 5 745 904 A (KIZER GEOFFREY MINARD ET AL) 28 April 1998 (1998-04-28) column 1, line 62 -column 2, line 35	5-8

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 99/09633

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9809238 A	05-03-1998	US 5897632 A	27-04-1999
EP 0420419 A	03-04-1991	JP 2559499 B	04-12-1996
		JP 3105544 A	02-05-1991
		CA 2025748 A	21-03-1991
		DE 69027827 D	22-08-1996
		DE 69027827 T	13-02-1997
US 5745904 A	28-04-1998	NONE	

This Page Blank (uspto)